AS Further Mathematics – Revision Notes
Unit D1 – Decision Mathematics 1

Algorithms

1. An algorithm is a precise set of instructions, which, if followed, will solve a problem. It must be unambiguous, and compute in a finite amount of time.
2. A flow chart can be used to represent an algorithm:
   a. Rectangular block – operation/instruction.
   b. Diamond block – question/decision with a yes/no answer.
   c. Oval block – terminus (start / stop).
3. **Bubble sort algorithm**
   a. Compare the first two items in the list. Swap them if they are in the wrong order.
   b. Move down the list until it has been passed through.
   c. Compare this list with that before the pass.
   d. If they are different, go back to the first step.
   e. If they are the same (no change) then stop – the list is sorted.
4. **Quick sort algorithm**
   a. Select the pivot of the list as its midpoint $\lceil \frac{1}{2}(n+1) \rceil$. Indicate this by ringing the pivot.
   b. Write all the numbers smaller than the pivot to its left and all those greater than or equal to it to its right.
   c. Apply the above steps to each sublist until they each contain one number.
5. For bin-packing, the minimum number of bins is given by $\left\lceil \frac{\sum h}{n} \right\rceil$.
6. For simple bin-packing problems, look for combinations that will fill a bin.
7. **First fit decreasing algorithm**
   a. Order the boxes in decreasing order of size using a sorting algorithm.
   b. Taking the boxes in this order, place the next box to be packed in the first available bin that will take that box.
8. This is a heuristic algorithm – it attempts to find a good solution, but does not necessarily give the optimum solution.
9. **Binary search algorithm**
   a. Sort the list into alphabetical order (ascending order for numbers).
   b. Take the middle item in the list as $\lceil \frac{1}{2}(n+1) \rceil$.
   c. If the middle item is less than the searched item, binary search the bottom half of the list.
   d. If the middle item is greater than the searched item, binary search the top half of the list.
   e. If the middle item is equal to the searched item, stop – the item has been found.
   f. If the list is empty, then stop – there is no match for the item.

Algorithms On Graphs

1. Graph and network definitions:

Graph (G)	A finite number of points (vertices or nodes) connected by lines (edges or arcs).
Path	A finite sequence of edges such that the end vertex of one edge is the start vertex of the next.
Cycle (circuit)	A closed path, i.e. the end vertex of the last edge is the start vertex of the first edge.
Hamiltonian cycle	A cycle that passes through every vertex of the graph once, and only once, and returns to its start vertex.
Eulerian cycle	A cycle that includes every edge of the graph exactly once (all valencies must be even).
Vertex set	Set of all the vertices of a graph
Edge set	Set of all the edges of a graph
Subgraph	A subset of the vertices together with a subset of the edges.
Connected graph	All pairs of vertices on the graph are connected (there is a path between each of them)
Simple graph	One in which there are no loops, i.e. no edges with the same vertex at each end, and not more than one edge connecting any pair of vertices.

Valency (degree or order) of a vertex	The number of edges connected to it.
Odd vertex	One with an odd valency
Even vertex	One with an even valency
Digraph	A graph in which the edges are directed (they have directions associated with them).
Tree	A connected graph with no cycles
Spanning tree	A subgraph of a graph G that includes all the vertices of G and is also a tree.
Complete graph	Every vertex is connected by an edge to every other vertex. K_n denotes a complete graph with n vertices.
Bipartite graph	Consists of two sets of vertices, X and Y. The edges only join vertices in X to vertices in Y, not vertices within a set. $K_{r,s}$ denotes a complete bipartite graph with r vertices in X and s vertices in Y.
Planar graph	No two edges meet one another, except at a vertex to which they are both incident, when the graph is drawn in a plane.
Isomorphic graph	Two graphs, G_1 and G_2 are isomorphic if they have the same number of vertices, and the degrees of the corresponding vertices are the same.
Network (weighted graph)	Each edge of the graph has a number (weight) associated with it. A network satisfies the triangle inequality if, for every triangle, no edge's weight exceeds the sum of the weights of the other two edges.

2. Graphs can be represented by:
    a. The vertex set and the edge set of the graph.
    b. An adjacency matrix – rows and columns of the matrix each represent a vertex, and the numbers in the matrix give the number of edges joining each pair of vertices (a loop is regarded as two edges). For a digraph, direction is required, so the columns and rows must be labelled 'from' and 'to' respectively.
3. A minimum spanning tree is a spanning tree such that the total length of its edges is as small as possible (sometimes called a minimum connector).
4. **Kruskal's MST algorithm**
    a. Sort the edges into ascending order of weight.
    b. Select the edge of least weight.
    c. Select from edges not previously selected the edge of least weight that does not form a cycle together with the edges already included.
    d. Repeat the above step until the selected edges form a minimum spanning tree.
5. Kruskal's algorithm is greedy (i.e. it makes an optimal choice at each stage) and it has drawbacks:
    a. The edges must be sorted to begin with.
    b. At each stage, a check that a cycle has not been formed must be made.
6. **Prim's MST algorithm (from a network)**
    a. Choose a starting vertex.
    b. Choose the vertex nearest the starting vertex and add the vertex and the corresponding edge to the tree.
    c. Connect to the tree the unconnected vertex that is nearest to any vertex in the tree.
    d. Repeat the above step until all vertices are connected.
7. **Prim's MST algorithm (from a distance matrix)**
    a. With the matrix representing the network, choose a starting vertex. Delete the row corresponding to that vertex.
    b. Label with '1' the column corresponding to the start vertex and ring the smallest undeleted entry in that column.
    c. Delete the row corresponding to the ringed entry.
    d. Label (with the next number) the column corresponding to the deleted row.
    e. Ring the lowest undeleted entry in all labelled columns.
    f. Repeat the last three steps until all rows are deleted. The ringed entries represent the edges in the minimum connector.
8. Complete enumeration can be used to find the shortest path between two vertices in a network, by listing all possible paths and comparing their lengths.
9. **Dijkstra's shortest path algorithm**
    a. Labels take the following form:

N (order)	V (label)
W (working values)	

   b. Label the start vertex $N = 1$ and $V = 0$.
   c. Update W for all vertices Y that can be reached directly from the vertex X that has just been labelled:
      New $W = W(X) + \text{weight}(XY)$
      New W replaces $W(Y)$ if new $W < W(Y)$ or if $W(Y)$ is null
   d. Out of all the unlabelled vertices with W not null, choose the vertex with the lowest value of W and label it with W and the next value of N.
   e. Repeat the above two steps until the destination vertex is labelled.
   f. The label of the destination vertex gives the length of the shortest path.
   g. To find the path, trace back through the network – if vertex N is on the path, vertex M is the previous vertex if the difference between the labels of M and N is equal to the weight of the edge MN.
10. **Planarity algorithm**
   a. Identify a Hamiltonian cycle in the graph.
   b. Redraw the graph so that the Hamiltonian cycle forms a regular polygon and all edges are straight lines within the polygon.
   c. Choose any edge PQ and decide this will stay in the polygon.
   d. Consider any edges that cross PQ – move these edges outside without producing any crossings. If it is not possible to do this then the graph is non-planar.
   e. Consider any remaining crossings inside, and see if any edge may be moved outside to remove it, without creating a crossing outside.
   f. Stop when all crossings have been considered – if there are no crossings inside or outside, then the graph is planar, otherwise it is non-planar.

The Route Inspection Problem

1. A traversable graph is one that can be drawn without removing the pen from the paper, and without going over the same edge twice.
2. A semi-Eulerian graph is one that allows a route starting and finishing at two different vertices, X and Y (as opposed to the same vertex for a Eulerian graph). This is only possible if X and Y are odd, and all the other vertices are even.
3. The Handshaking theorem:
   a. $\sum (\text{valencies}) = 2\,(\text{number of edges})$
   b. $\sum (\text{odd valencies}) + \sum (\text{even valencies}) = 2\,(\text{number of edges})$
   c. As the sum of the even valencies must be even, the sum of the odd valencies must also be even. This means that there will always be an even number of odd vertices.
4. The route inspection problem is that of finding a route of minimum weight that traverses every edge at least once, returning to its starting vertex.
5. **Route inspection algorithm**
   a. List all odd vertices.
   b. Form all possible pairings of odd vertices.
   c. For each pairing, find the edges that are best to repeat, and calculate the sum of the weights of these edges.
   d. Choose the pairing with the smallest sum. Construct a Eulerian cycle that repeats these edges.

Critical Path Analysis

1. A complex project is broken down into sub-projects or activities. They are not all independent, so the dependencies between activities are shown using a precedence (or dependence) table (this only shows activities dependant upon those that immediately precede them).
2. An activity network models a project, based on a precedence table:
   a. The nodes represent events (the completion of one or more activities).
   b. The arcs represent activities (the weight represents the duration of the activity).
   c. The source node is the start of the project, and the sink node is the end.
   d. Arrows are used to make the directionality clear (usually time flows from left to right).
3. The event at the beginning of an activity is its tail event, and that at the end is its head event. An activity can be represented as (T, H), where H > T.
4. A dummy activity has zero duration, and is shown by a dotted line. It indicates that all activities dependant on its head event cannot take place until its head event has taken place.
5. The critical path is the longest path from the source to the sink node.

6. **Critical path algorithm**
    a. Forward scan (earliest event time e_i):
        i. This is the earliest time that you can arrive at an event i with all the incoming activities completed. Work forwards on the diagram.
        ii. $e_i = \max[e_k + \text{duration}(k, i)]$ where (k, i) are all arcs leading into i.
        iii. The earliest event time of the sink node is the length of the critical path.
    b. Backward scan (latest event time l_i):
        i. This is the latest time that you can leave event i without extending the length of the critical path. Work backwards on the diagram.
        ii. $l_i = \min[l_k - \text{duration}(i, j)]$ where (i, j) are all arcs leaving i.
7. Critical events and activities:
    a. The slack of an event is the difference between the latest event time and the earliest event time. A critical event is one with zero slack.
    b. The total float of an activity is given by $F(k, i) = [l_i - e_k - \text{duration of } (k, i)]$. A critical activity is one with zero float (i.e. it lies on the critical path).
8. Activity times (for an activity (i, j)):
    a. Earliest start time (e_i).
    b. Earliest finish time (e_i + length (i, j)).
    c. Latest finish time (l_j).
    d. Latest start time (l_j – length (i, j)).
    e. Total float = latest finish – earliest finish, or latest start – earliest start.
9. A Gantt diagram (or cascade diagram) can be used to display this information:
    a. Critical activities are drawn along the top (zero float).
    b. Other activities have boundaries indicated by dotted arrows, and the activity is shown (as bars) as early and as late as possible.
    c. Critical events are indicated (by circled numbers) at the junctions of critical activities.
10. Scheduling of activities:
    a. No worker may remain idle if there is an activity that can be started.
    b. Once a worker starts an activity, he must continue until it is finished.
    c. When a worker completes an activity, consider all activities that have not been started but may now be started.
    d. Assign the worker to the activity whose latest start time is smallest (i.e. the most critical).
    e. If there are no activities that can be started, the worker will have to wait until an activity can be assigned.

Linear Programming

1. Formulating a problem:
    a. Decision variables – the quantities you need to know to solve the problem.
    b. Constraints – limits on resources and/or relationships between decision variables. Non-negativity constraints are when the decision variables must be ≥ 0.
    c. Objective function – the function of decision variables to be optimised
2. Graphical solutions (2 variable):
    a. All the inequalities for the constraints are drawn on the same graph.
    b. The admissible set is the area that satisfies the inequality (shown by arrows from the line).
    c. The inadmissible points are in the area that doesn't satisfy the inequality, shown by shading lines.
    d. The region into which all the arrows point is the feasible region, and contains all the feasible solutions to the problem.
3. The ruler method (objective line method) of optimisation:
    a. Draw the constraints and the feasibility region.
    b. Draw a member of the family of lines $P = \alpha x + \beta y$.
    c. Move your ruler parallel to this until it loses contact with the feasible region.
    d. Find the coordinates of this point using simultaneous equations.
    e. Put these into the equation to get a value for P.
4. The vertex method of optimisation:
    a. Obtain the coordinates for all the vertices of the feasibility region.
    b. Evaluate the objective function at each of these vertices.
    c. The optimal value of the function will be the maximum or minimum of these values, and the coordinates that it occurs at give the values of x and y for the solution.

5. Integer valued solutions (where the decision variables must be integers):
    a. Solve the problem using the ruler or vertex method.
    b. Find points close to the optimal point that have integer coordinates in the feasible region.
    c. Calculate P for each of these coordinates, and choose the optimal value.
6. Inequalities can be converted into equations by adding a slack variable – this gives the difference between the amount of resource available and the amount used.
7. At a vertex of the feasible region, two of the variables x, y, s and t are zero.
8. For x = number of variables – number of equations:
    a. A basic solution is when x variables are set to zero and the equations solved.
    b. Non-basic variables are those set to zero.
    c. Basic variables are those that are solved.
    d. A basic feasible solution is a basic solution that is feasible.
9. A linear programming problem is in standard form if:
    a. The objective function $\alpha x + \beta y + \gamma z$ is to be maximised.
    b. All the constraints (other than non-negativity) are of the form $ax + by + cz \leq d$.
10. A simplex tableau is a table used for the simplex method. The initial tableau will be of the form:

Basic variable	x	y	z	r	s	**Value**
r	4	5	3	1	0	16
s	5	4	6	0	1	24
P	-14	-12	-13	0	0	0

Maximise $P = 14x + 12y + 13z$
Subject to $4x + 5y + 3z \leq 16$
$5x + 4y + 6z \leq 24$

11. The optimality condition – if the objective row of a tableau has zero entries in the columns labelled by basic variables, and no negative entries in the columns labelled by non-basic variables, then the solution represented by the tableau is optimal.
12. **The Simplex algorithm**
    a. Choose the pivotal column (the non-basic variable to become basic) by choosing that with the most negative entry in the objective function row. Show this column with an arrow.
    b. Choose the pivotal row (the basic variable to leave the basis) by calculating θ values for each row other than the objective row (the entry in the pivotal column must be positive). The row with the smallest θ value is the pivotal row (indicated by an arrow):
$$\theta = \frac{\text{entry in value column}}{\text{entry in pivotal column}}$$
    c. The pivot is at the intersection of the pivotal row and column – it should be ringed. Divide all entries in the pivotal row by the pivot.
    d. Add multiples of the new pivotal row to all other rows so that all other elements in the pivotal column become zero.
    e. Repeat the above steps until the optimal tableau is formed.

Matchings

1. In drawing a bipartite graph, use open circles for one vertex set, and solid circles for the other. It can also be shown as a matrix, with columns for vertex set Y and rows for vertex set X.
2. A matching in a bipartite graph G is a subset M of the edges E of the graph such that no two edges in M have a common vertex:
    a. Trivial matching – one in which there are no edges.
    b. Non-trivial matching – a matching with edges.
    c. Maximal matching – a matching in which the number of edges is as large as possible. The maximum number of edges is equal to the number of vertices in the smallest set.
    d. Complete matching – in a bipartite graph with n vertices in each set, a complete matching is a matching in which the number of edges is also n.
3. An alternating path for M in G is a path in G for which:
    a. It joins an unmatched vertex in X to an unmatched vertex in Y.
    b. The edges in the path are alternately in and not in the matching M.
    c. The number of edges not in M is one more than the number of edges in M.
4. Breakthrough is achieved when an unmatched vertex in the second set is reached.
5. To find an alternating path between X and Y:
    a. Choose an unmatched vertex in set X.
    b. Draw vertices joined to this on the graph, as a tree, joined to the first vertex.

   c. If these are matched, join the matchings on the tree.
   d. Add any vertices joined to these onto the tree (if not already on the tree).
   e. Repeat until breakthrough is reached (do not add vertices already on the tree).
6. **Matching improvement algorithm**
   a. Start with any non-trivial matching M in G.
   b. Search for an alternating path for M in G.
   c. If an alternating path is found, construct a better matching M' by changing the status of the edges in the alternating path. Repeat the previous step.
   d. Stop when no alternating path can be found – the matching obtained is maximal (if a matching M in a bipartite graph G is not a maximal matching, then G contains an alternating path for M).

Flows In Networks

1. A capacitated network is a weighted digraph, in which the weights are the capacities of the arcs – indicating the maximum flow that each can carry:
   a. A vertex S is a source if all arcs containing S are directed away from S.
   b. A vertex T is a sink if all arcs containing T are directed towards T.
2. A flow in a network N with a single source S and a single sink T is an assignment to each arc *e* of a non-negative number called the flow along the arc *e*. This must satisfy:
   a. Feasibility condition – the flow along each arc cannot exceed the capacity of the arc.
   b. Conservation condition – sum of flows on arcs into V = sum of flows on arcs out of V. Likewise, sum of flows out of S = sum of flows into T (this is the value of the flow).
3. If the flow along the arc *e* is equal to its capacity, then the arc is saturated – otherwise it is unsaturated.
4. A zero flow is when the flow of every arc is zero. Any other flow is a non-zero flow.
5. When drawing capacitated networks, the flows are placed in circles, and the capacities are the weights of the arcs.
6. Arc labelling procedure – each arc is labelled with two numbers:
   a. The remaining excess capacity of the arc, x (shown with a forward arrow) – the amount by which the flow along the arc may be increased.
   b. The flow along the arc, y (shown by a backward arrow) – the back capacity or amount by which the flow may be reduced.
   c. Total capacity for an arc = $x + y$.
7. Flow-augmenting paths are a series of unsaturated paths that, when put together, give a maximal flow through the network. To find a flow-augmenting path:
   a. Look for a path that does not include any saturated edges (all edges have excess capacity greater than zero).
   b. An existing flow can be decreased, by going back along the arc, as part of the flow-augmenting path. This is a back flow
   c. A flow-augmenting path consists of:
      i. Forward arcs – unsaturated paths directed along the arc.
      ii. Backward arcs – arcs directed against the direction of the path (non-zero flow).
8. A cut in a network, with source S and sink T, is a set of arcs A whose removal separates the network into two parts, X and Y, whereby S is within X and T within Y:
   a. A cut's capacity is the sum of the capacities of the arcs in the cut directed from X to Y.
   b. A minimum cut is a cut of the smallest possible capacity.
9. The max flow - min cut theorem:
   a. The value of the maximal flow = the capacity of a minimum cut.
   b. A flow through a network is maximal if and only if a cut can be found with capacity equal to the value of the flow.
10. **Maximum flow algorithm**
   a. Obtain an initial flow by inspection.
   b. Find flow-augmenting paths using the labelling procedure until no further flow-augmenting paths can be found.
   c. Check that the flow obtained is maximal by finding a cut with capacity equal to the value of the flow (max cut - min flow theorem).
11. If there are multiple sources, introduce a supersource S linking to sources S_1 and S_2. The capacity of SS_1 is the sum of the capacities leaving S_1. For a supersink T, the capacity of T_1T is the sum of the capacities entering T_1. Find the maximum flow for this network, then remove the supersource and supersink to give the maximum flow of the original network.